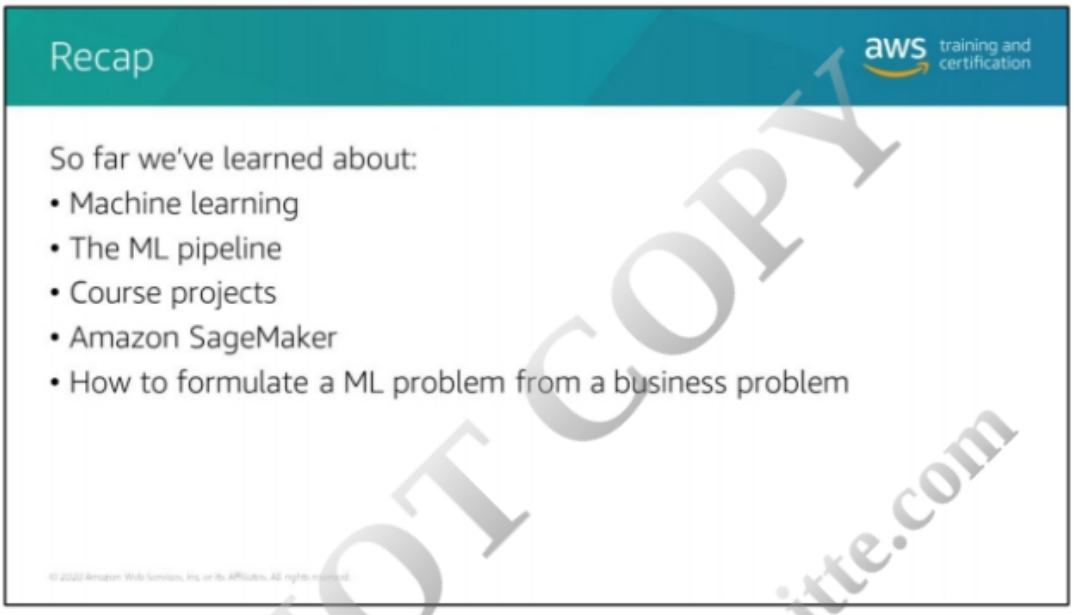
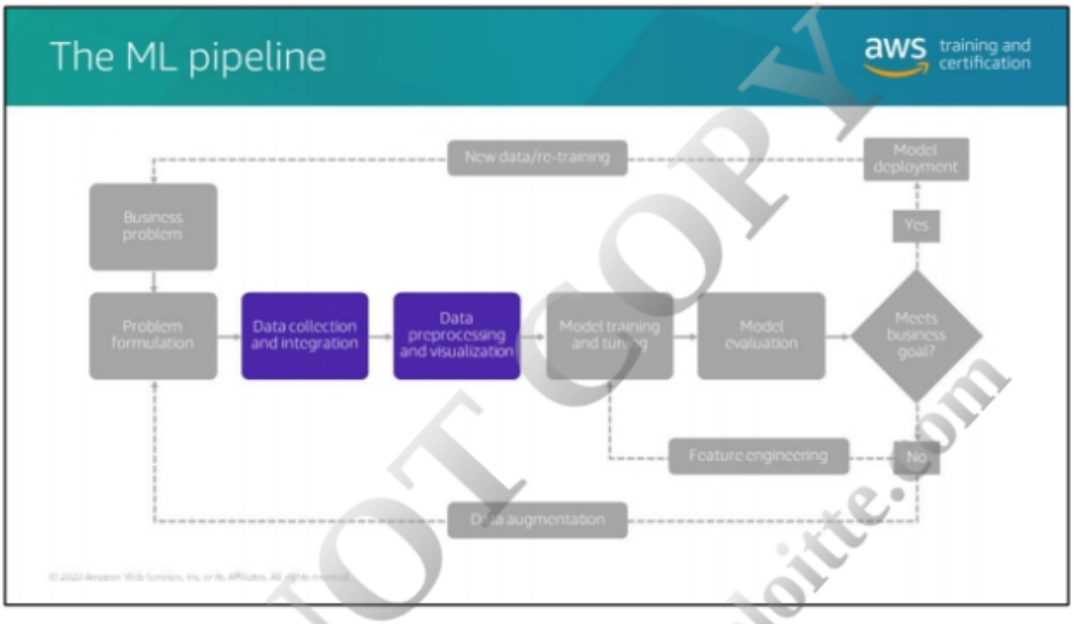


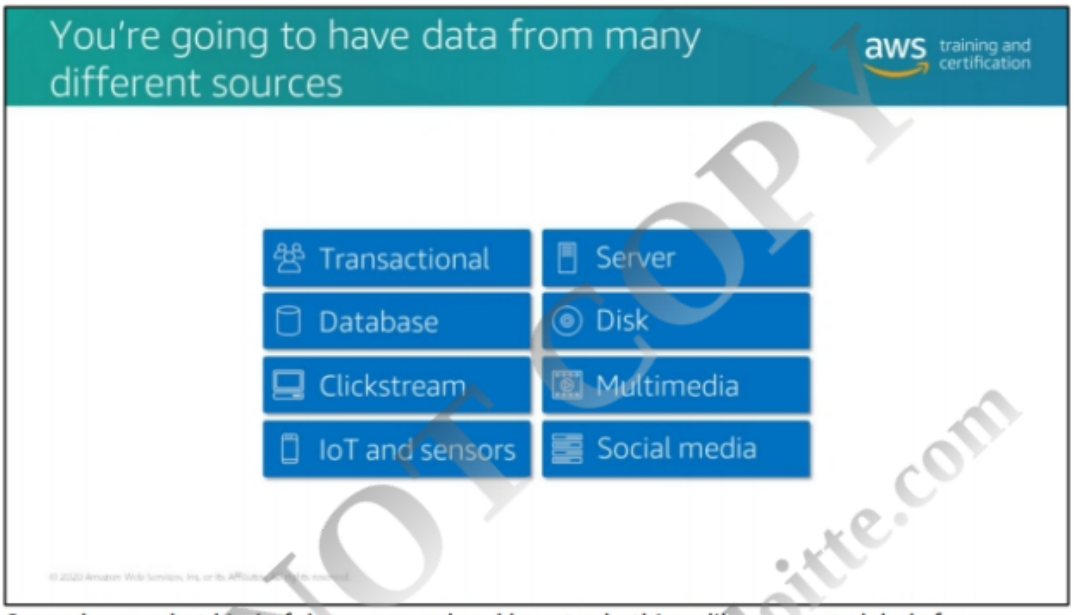
Navigate here to access Checkpoint #1:
https://amazonmr.au1.qualtrics.com/jfe/form/SV_7aH7TW8Dwndo9gN



Now that we've explored how to turn business problems into ML problems and have a better understanding of how to define success metrics, examine and choose your data, and choose your algorithm – it's time to start getting that data ready. This module covers how to preprocess your data and get it ready for training your model.

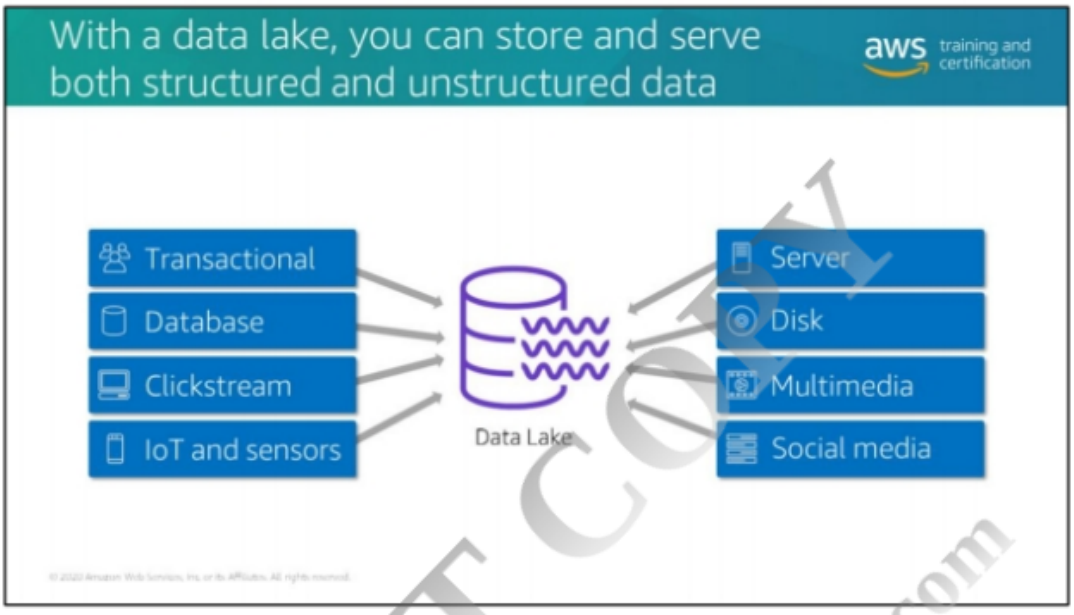
This module will focus on the next stage of the ML pipeline: Data preprocessing. This module covers data collection, reformatting data and techniques for cleaning up dirty data.

Note here that data visualization is woven throughout this module as, in reality, visualization and various preprocessing techniques happen simultaneously throughout this phase of the pipeline.



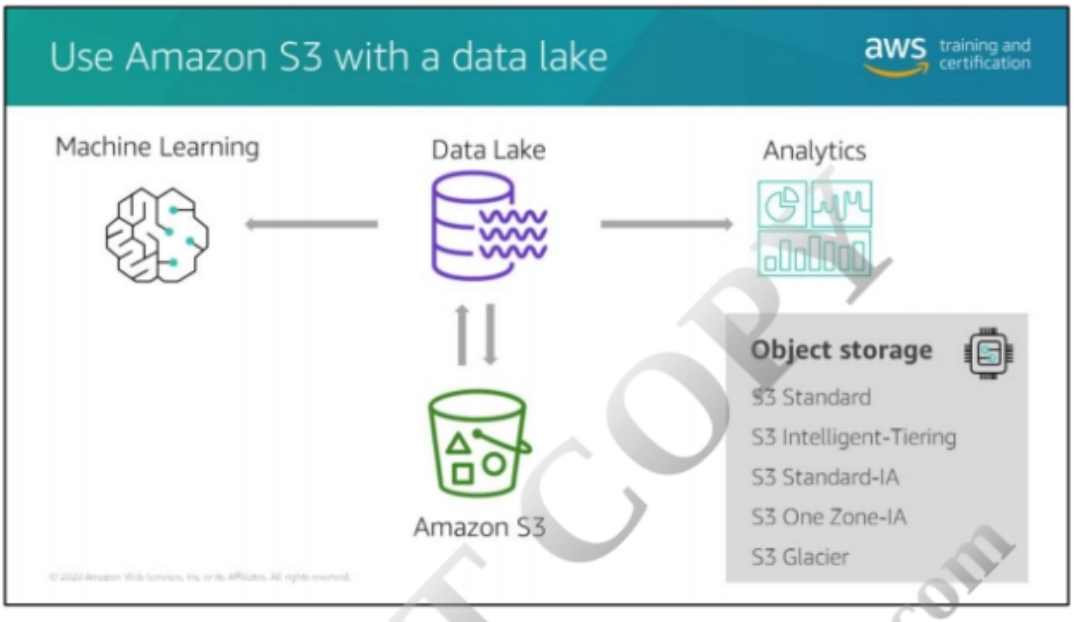
So we know what kind of data we need and how to do things like generate labels for datasets that aren't already labeled. Next, the question becomes: do we have access to this ideal data, what form is it in, and where is it currently stored? In real life, ML projects have data coming in from multiple sources and representing different types of data, including structured, semi-structured, and unstructured data. Data can come from a CSV file or a traditional database, for instance. When using the AWS Cloud, it could come from AWS storage services like Amazon Redshift, Amazon DynamoDB, Amazon RDS, or Amazon Glacier.

Organizations are often challenged to access and analyze their data when its stored in different formats across various, disparate locations. With data continuously being collected from a variety of sources, if not properly addressed, this challenge only grows bigger as organizations age and grow. A lack of simplified access to data creates workflow bottlenecks as employees often need to request help from IT to access the information required to build, train, and deploy machine learning models. To address these challenges, organizations are forced to seek out a solution that offers a single source of truth, that is readily accessible when, and to the employees that need it. This is often difficult and costly to maintain as an on-premises solution.



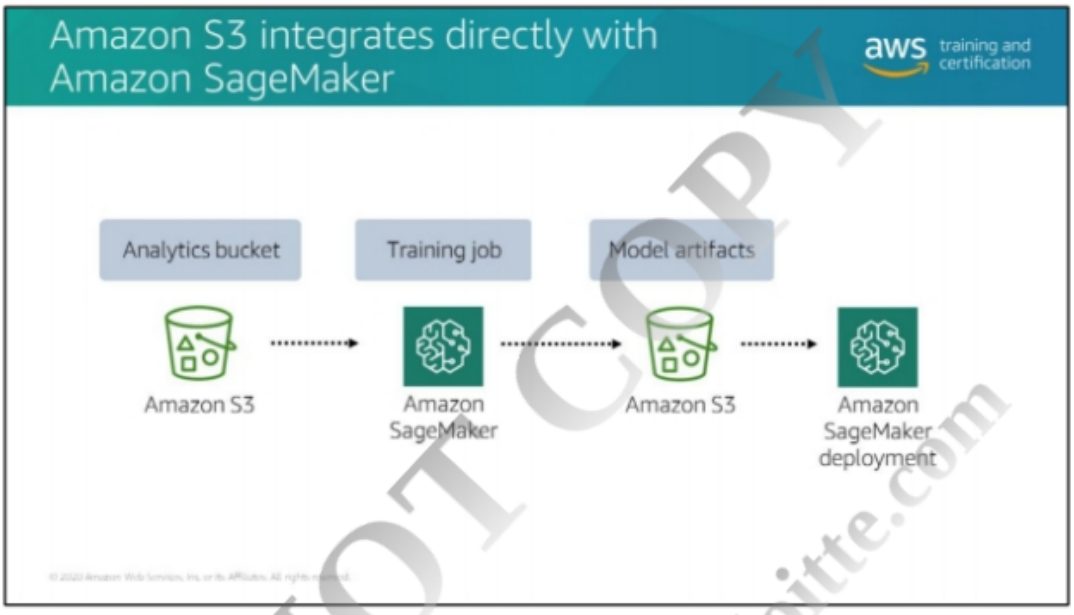
A data lake architecture can provide a solid foundation on which to build a solution to this challenge. A data lake allows you to store massive amounts of data in a central repository so it's readily available to be categorized, processed, enriched, and consumed by diverse groups within an organization.

There's no silver bullet to building a data lake, however. In most cases, building a data lake entails the use of dozens of technologies, tools, and environments, including data from third parties. When done right, a data lake can open the door to a whole new set of advanced analytics, facilitating data science and machine learning.



The data lake foundation is designed to provide you with a single source of truth, with additional features including data submission, ingest processing, aggregation, analysis, and searching capabilities.

Amazon S3 is the preferred storage option with a data lake for data science processing on AWS. Amazon S3 provides highly durable storage and seamless integration with various data processing services and ML platforms on AWS. It can be used as "one source of truth" storage for most AWS ML services.



So why should you use Amazon S3 to address your machine learning needs?

First, you can use Amazon S3 to store your training datasets that you will use for your machine learning models. These files can be delivered directly to your Amazon SageMaker training jobs from Amazon S3.

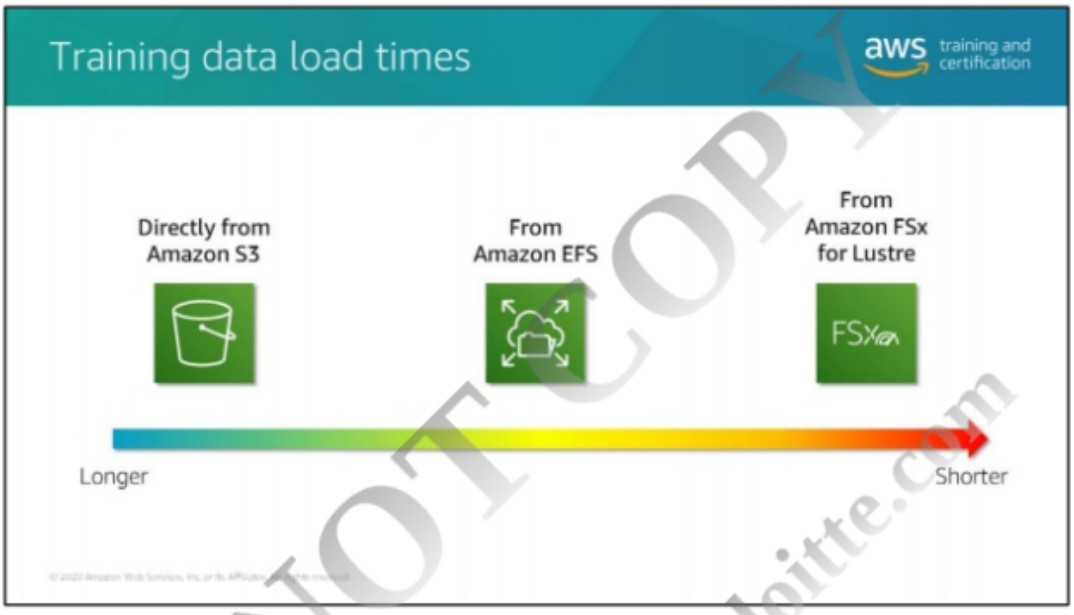
Second, Amazon S3 can also be used for storing objects like trained ML models or images or videos that need preprocessing before predictions including workloads with large amounts of user generated content, such as video or photo sharing.

Alternatively, if your training data is already in Amazon Elastic File System (Amazon EFS), we recommend using that as your training data source. Amazon EFS has the benefit of directly launching your training jobs from the service without the need for data movement, resulting in faster training start times. This is often the case in environments where data scientists have home directories in Amazon EFS and are quickly iterating on their models by bringing in new data, sharing data with colleagues, and experimenting with including different fields or labels in their dataset.

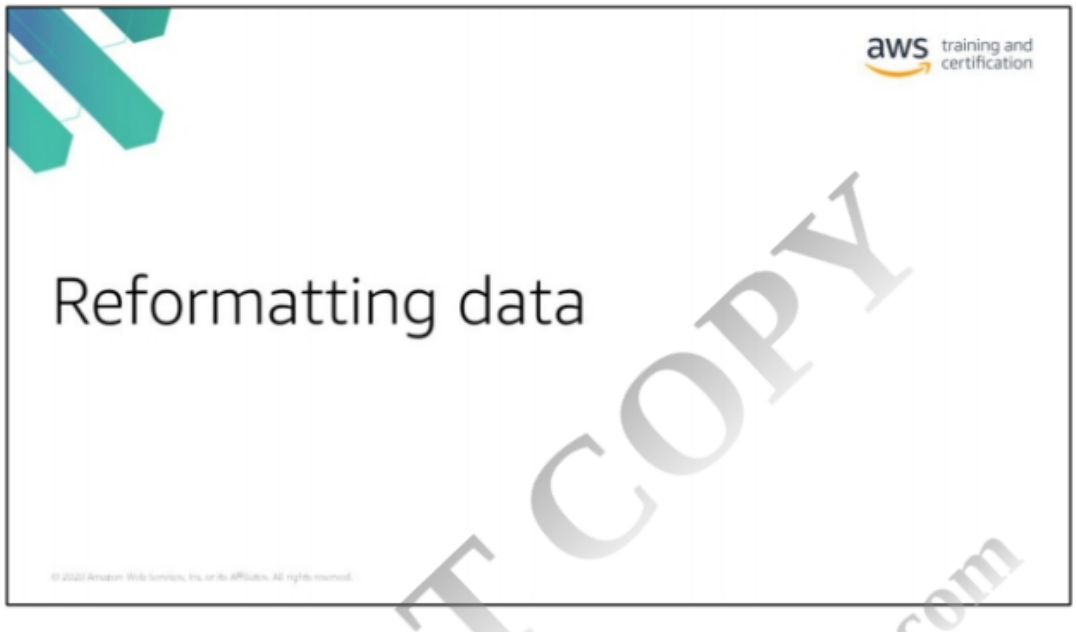
For example, a data scientist can use a Jupyter notebook to do initial cleansing on a training set, launch a training job from Amazon SageMaker, then use their notebook to drop a column and re-launch the training job, comparing the resulting models to see which works better.

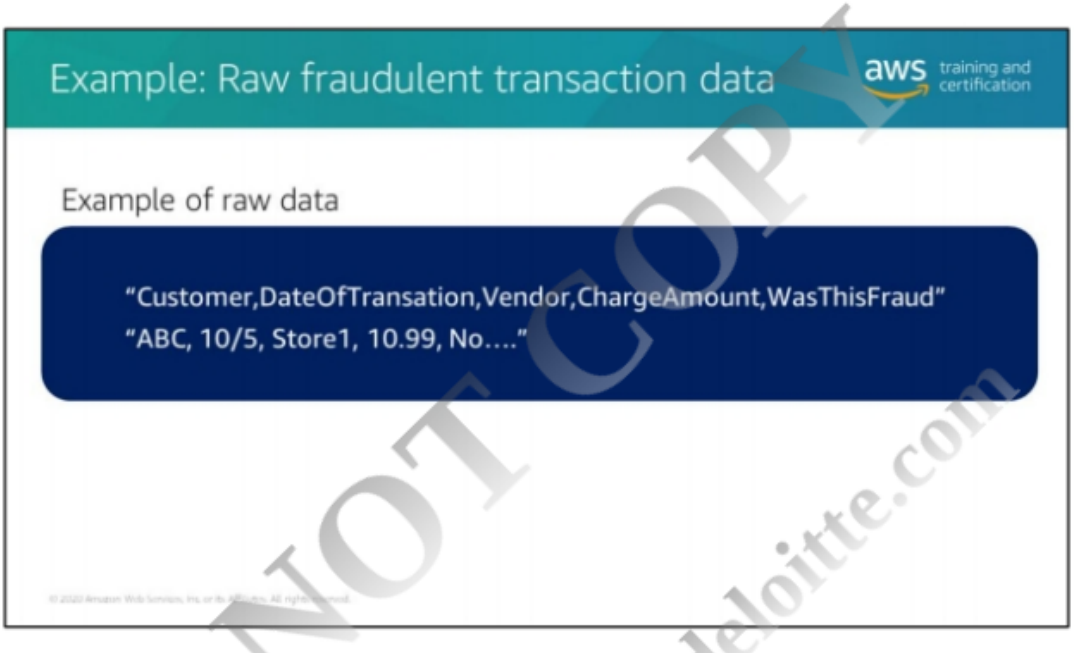
When your training data is already in Amazon S3 and you plan to run training jobs several times using different algorithms and parameters, consider using Amazon FSx for Lustre, a file system service. FSx for Lustre speeds up your training jobs by serving your Amazon S3 data to Amazon SageMaker at high speeds. The first time you run a training job, FSx for Lustre automatically copies data from Amazon S3 and makes it available to Amazon SageMaker. You can use the same Amazon FSx file system for subsequent iterations of training jobs, preventing repeated downloads of common Amazon S3 objects.

You can shut down Lustre whenever your workloads complete, and only use it during training.



When it comes to running ML workloads, you want to be cognizant of the training load time. Consider loading images, for example, if you were to do a quick comparison of Amazon S3 and Amazon EFS, you'd notice that Amazon S3 is the cheaper option for data storage. Storing the image datasets which are large in size in to Amazon S3` makes sense when you don't have a training job time constraints that you need to take into consideration. When you have training job time constraints, you want your data to be as close to instances as possible while providing the high throughput and sub-second latencies. You can use Amazon FSX for Lustre and Amazon EFS for these types of workloads.



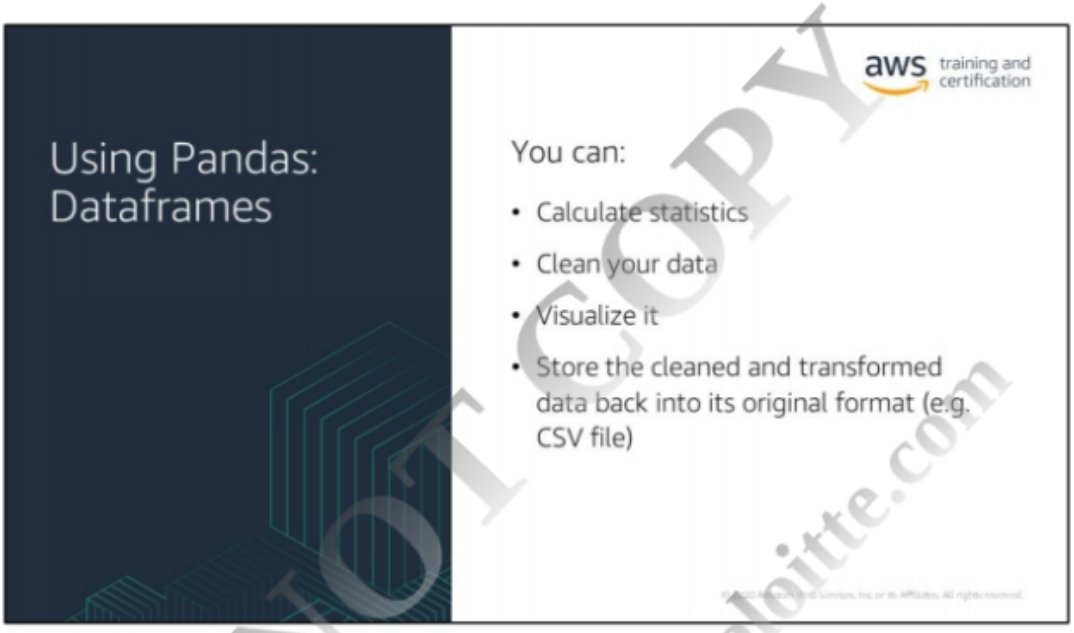


Think back to our fraudulent transaction scenario and take a look at the example on the screen. Before you can start running statistics on your data to better understand what you're working with, you have to ensure it's in the right format to be analyzed. So how do you get your data into the proper format? You should begin by asking yourself questions like these:

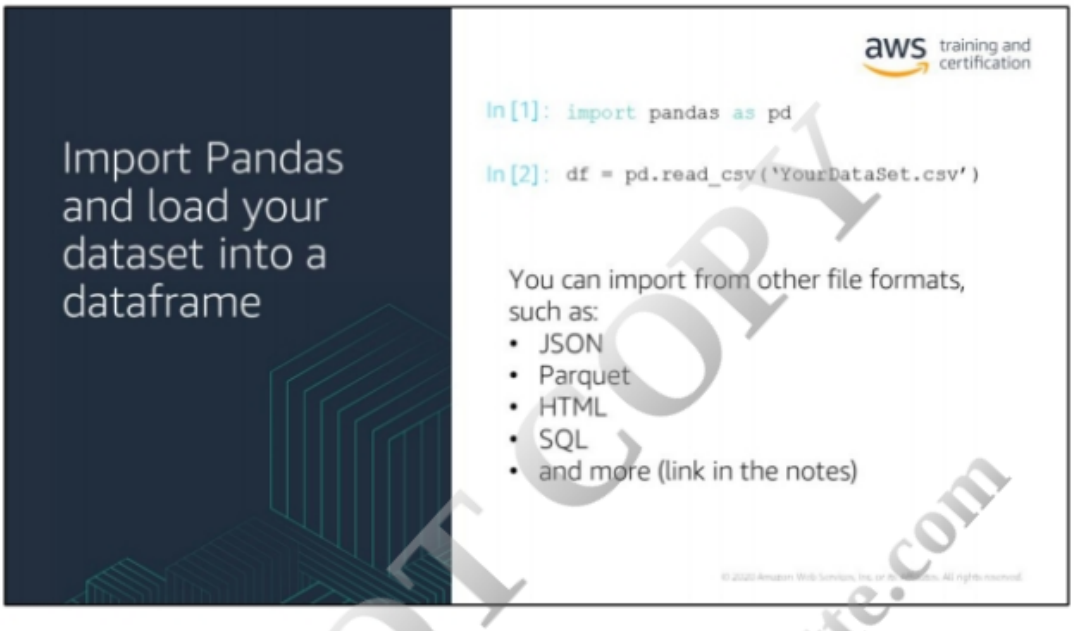
- How would I fix the problem with this data?
- What tools should I use to help me fix it?

Pandas, an open source Python library, can be used for this particular data reformatting. Pandas reformats data from various formats like CSV, JSON, Excel, Pickle, and others into a tabular representation, presenting it in rows and columns.

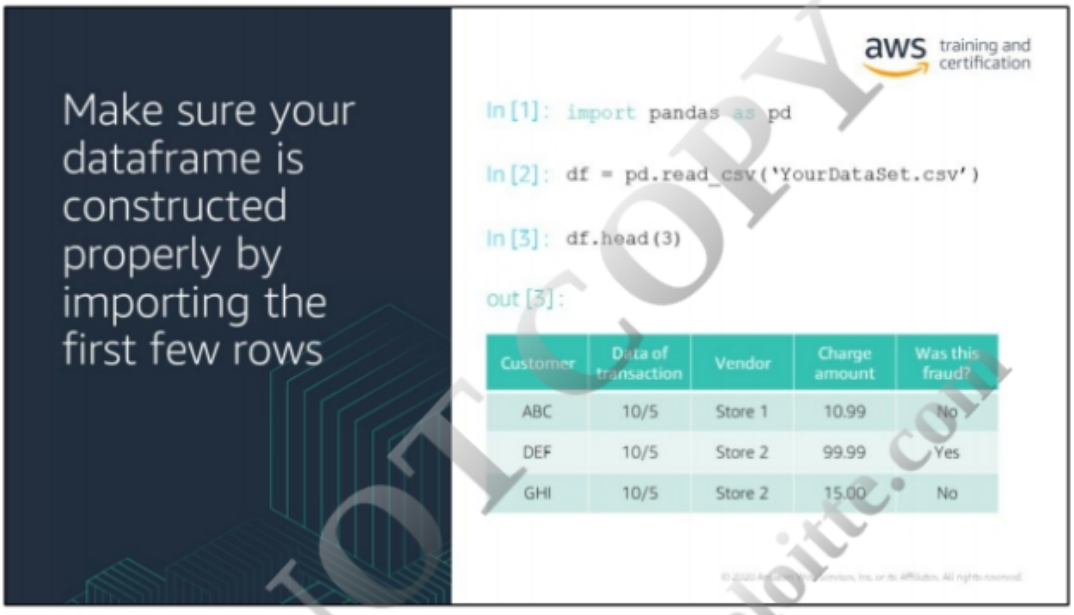
The tabular format that Pandas puts your data into is called a *dataframe*. A dataframe is made up of many *series*, which are essentially columns capable of holding any data type. The axis labels are referred to as the *index*.



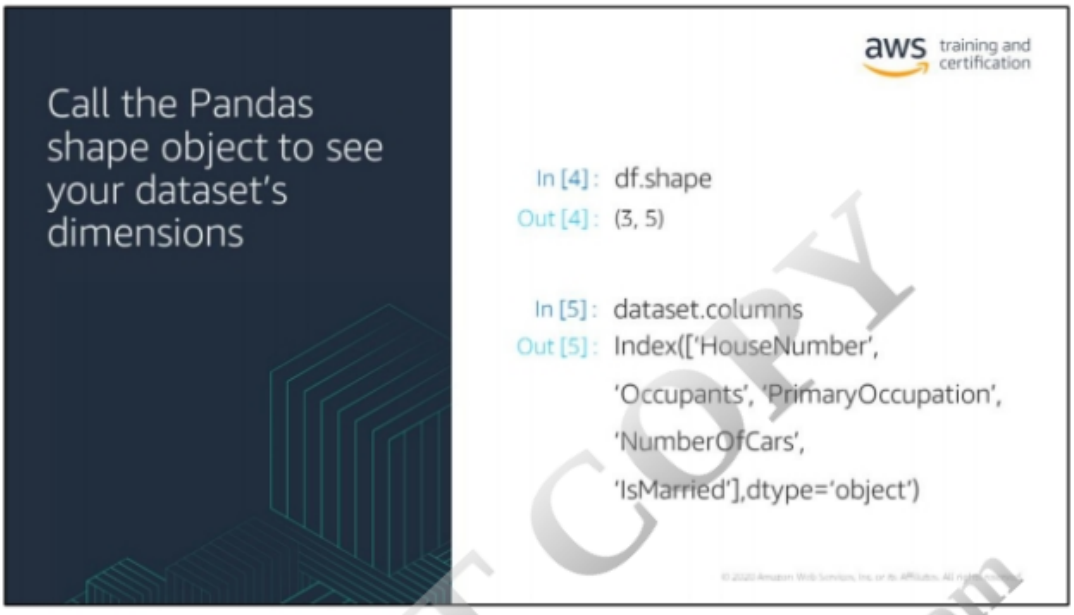
With your data in a Pandas dataframe, you can calculate statistics, clean your data, visualize it, and even store the cleaned and transformed data back into its original format (e.g., a CSV file).



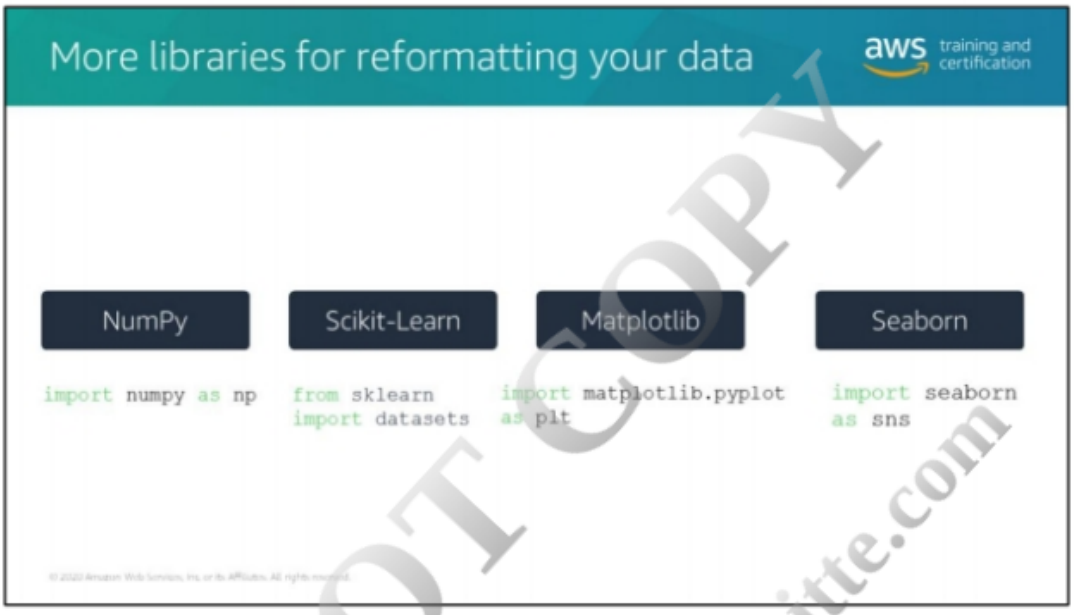
To get started, you can import Pandas using the code above. Then, import the dataset from a local CSV file, for instance, and then show the first few rows of the dataframe. You can also import from other file formats. The complete list of compatible file formats is here: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html



Then, import the dataset from a local CSV file, for instance, and then show the first few rows of the dataframe.

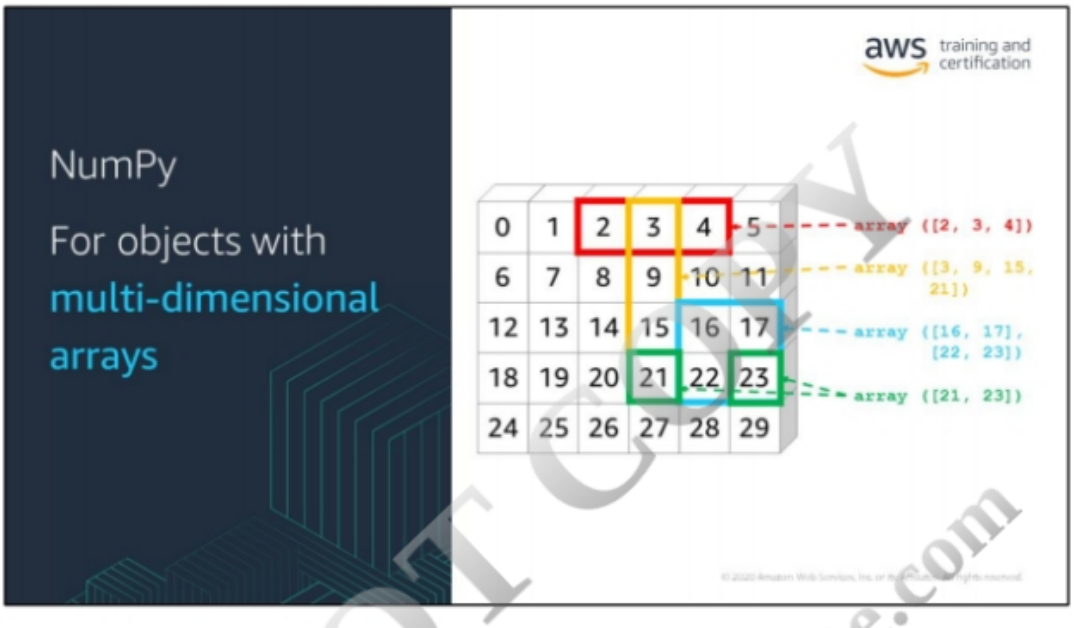


With Pandas, for instance, you can call the shape object to see your dataset's dimensions. Shape returns the number of rows and columns. Columns returns the names and types of your columns, and Rows does the same for rows.



In machine learning, libraries are tools containing a premade set of routines and functions that enable you to format different types of data, like numerical or image data.

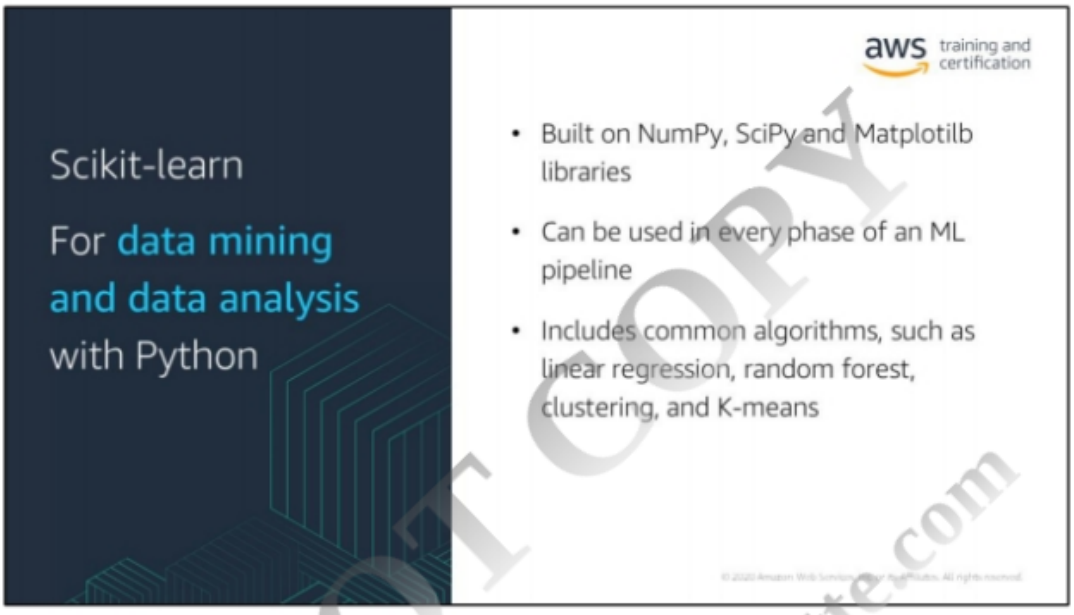
Here's an overview of common tools used at this preprocessing phase of the ML pipeline.



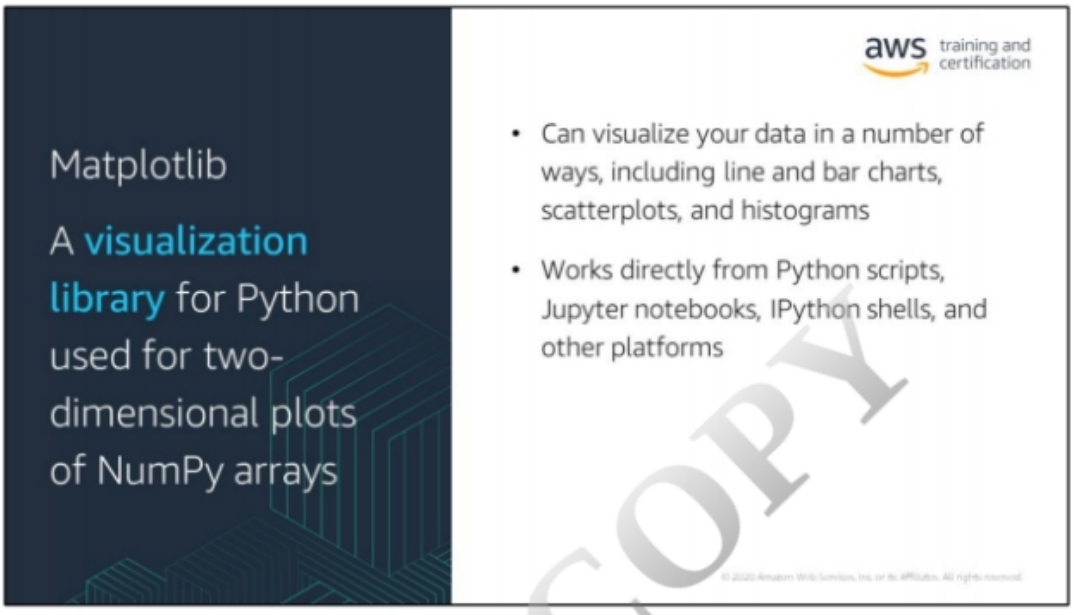
NumPy: NumPy is a Python library that serves as a fundamental package for scientific computing while working with Python. It's a general-purpose array-processing package that provides high-performance multidimensional array object and the tools for working with these arrays. Its features include:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, and random number capabilities

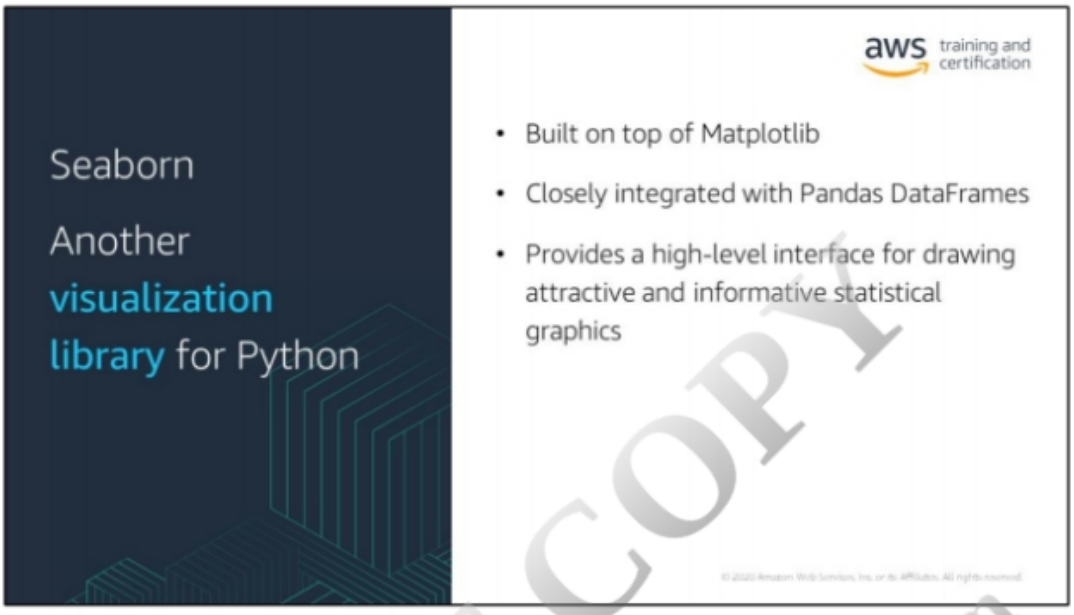
Given its features, NumPy is a common tool used at the preprocessing phase of the ML pipeline as it can help you reformat your data and run a variety of statistics on that data to better understand it.



Scikit-learn: Scikit-learn is an open source Python library that includes various tools for data mining and data analysis. It is built on NumPy, SciPy and Matplotilb libraries. It can be used in every phase of a machine learning pipeline including data preprocessing. We'll circle back to Scitkit-learn during the model training phase of the pipeline as well, as the library includes supervised algorithms like linear regression and random forests as well as unsupervised algorithms like clustering and K-means.



Matplotlib: Matplotlib is a visualization library in Python used for two-dimensional plots of NumPy arrays. With Matplotlib you can visualize your data in a number of ways, including line and bar charts, scatterplots, and histograms. You can leverage the library directly in your Python scripts, Jupyter notebooks, iPython shells, and other platforms.



Seaborn: Seaborn is another Python data visualization library. It's built on top of Matplotlib and is closely integrated with Pandas DataFrames. It provides a high-level interface for drawing attractive and informative statistical graphics.

Now that you have reformatted your data, it's time to begin cleaning it up.

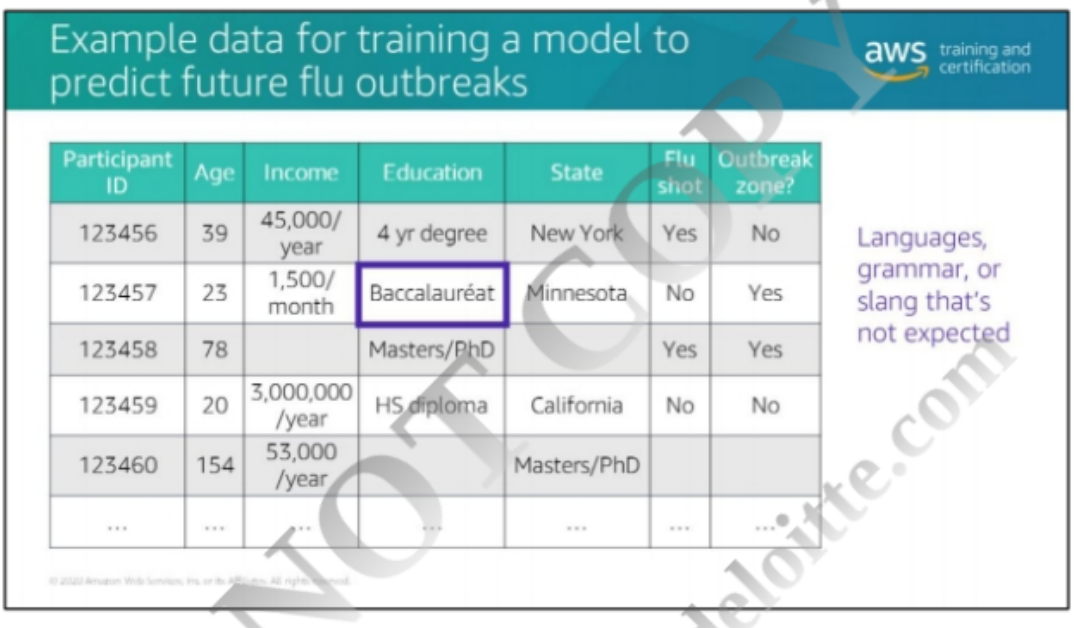
Example data for training a model to predict future flu outbreaks

aws training and certification

Participant ID	Age	Income	Education	State	Flu shot	Outbreak zone?
123456	39	45,000/ year	4 yr degree	New York	Yes	No
123457	23	1,500/ month	Baccalauréat	Minnesota	No	Yes
123458	78		Masters/PhD		Yes	Yes
123459	20	3,000,000 /year	HS diploma	California	No	No
123460	154	53,000 /year		Masters/PhD		
...	...	...	...	...	...	...

For the purposes of demonstrating how we're going to pre-process some data, here is an example of a few fictional rows of data we've created for the purposes of this course. This dataset would be used to predict future flu outbreaks by training the model on historical data about people, including demographic data as well as whether they got a flu shot last year, along with some corresponding data about their location and whether it was subject to a flu outbreak during that year's flu season.

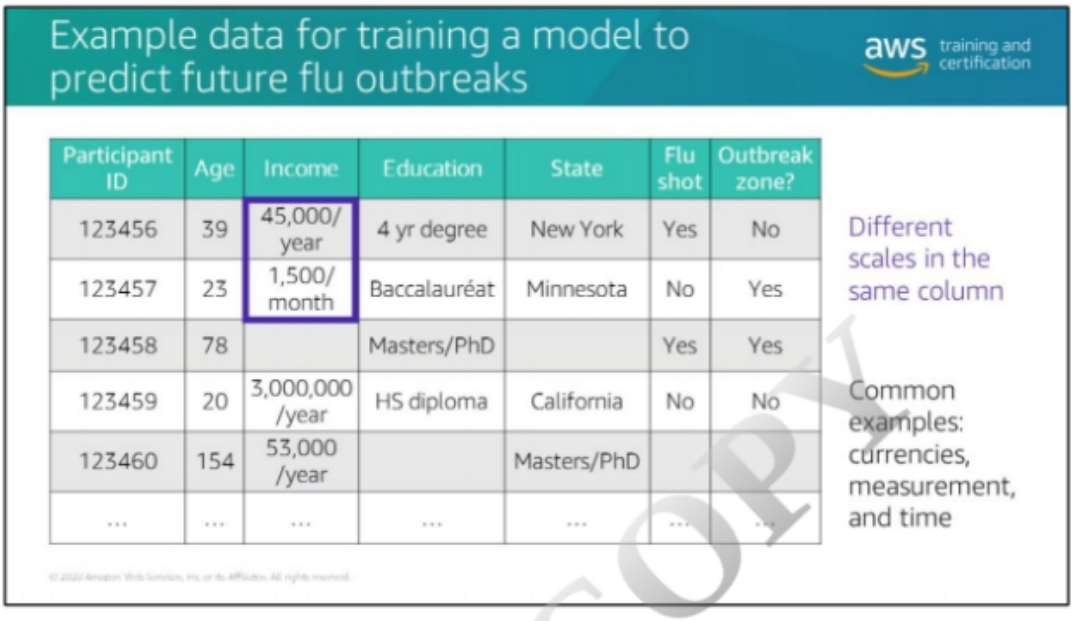
Now the problem with this dataset is pretty clear. We've got a problem with what's called *dirty data*.



Participant ID	Age	Income	Education	State	Flu shot	Outbreak zone?
123456	39	45,000/ year	4 yr degree	New York	Yes	No
123457	23	1,500/ month	Baccalauréat	Minnesota	No	Yes
123458	78		Masters/PhD		Yes	Yes
123459	20	3,000,000 /year	HS diploma	California	No	No
123460	154	53,000 /year		Masters/PhD		
...	...	...	...	...	...	...

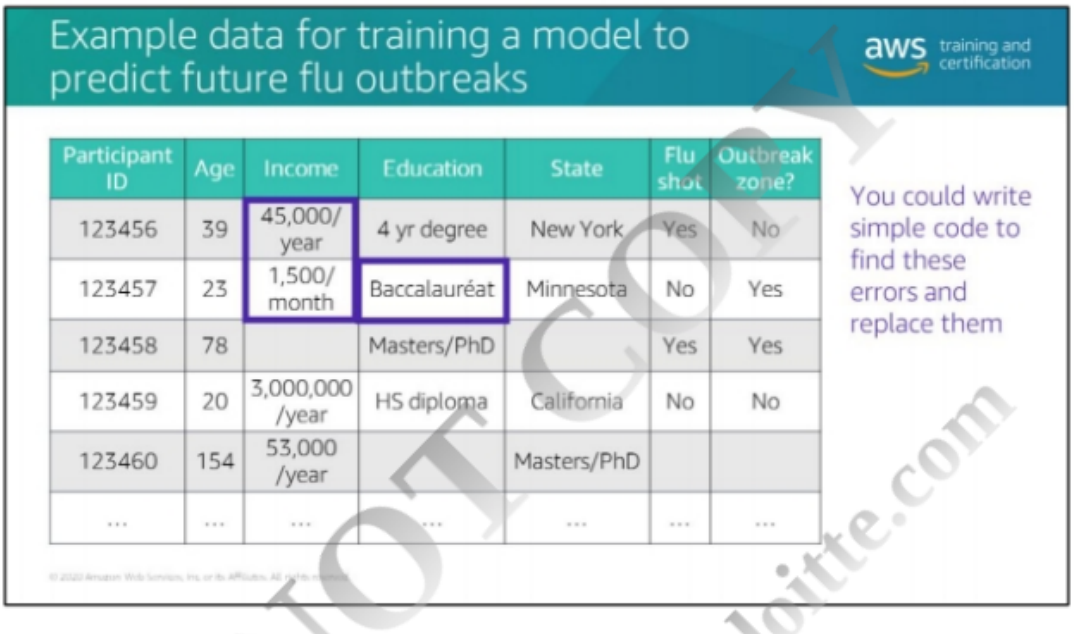
Data can be messy in several ways. For instance, maybe your algorithm expects to see data written in English, but there are some words in your data set from different languages. Or maybe there are special characters in some of the words, or even just a lot of space in between words. The key is to make sure you are standardizing your data. If your algorithm requires English, make sure it's all in English.

The same goes for grammatical structure. For example, convert your text data into all lowercase so the same word isn't treated as two different words just because of its capitalization.



Participant ID	Age	Income	Education	State	Flu shot	Outbreak zone?
123456	39	45,000/ year	4 yr degree	New York	Yes	No
123457	23	1,500/ month	Baccalauréat	Minnesota	No	Yes
123458	78		Masters/PhD		Yes	Yes
123459	20	3,000,000 /year	HS diploma	California	No	No
123460	154	53,000 /year		Masters/PhD		
...	...	...	...	...	...	...

Your data set might also include data that is on very different scales. For example, here we have one column called *Length*, but that column has different units for data, like kilometers, meters, and miles. This is a common occurrence in many numerical data sets, especially if your dataset is a result of merging data from multiple sources.



Participant ID	Age	Income	Education	State	Flu shot	Outbreak zone?
123456	39	45,000/ year	4 yr degree	New York	Yes	No
123457	23	1,500/ month	Baccalauréat	Minnesota	No	Yes
123458	78		Masters/PhD		Yes	Yes
123459	20	3,000,000 /year	HS diploma	California	No	No
123460	154	53,000 /year		Masters/PhD		
...	...	...	...	...	...	...

For both of these kinds of examples, once you identify common errors like these, you can write some simple code that finds these errors and changes them. For instance, your code could look for anything in the income column that says "/month" or "/per month" and multiply the value by 12. Or for unexpected languages, it could replace the entries with the word in the expected language for your data.

Example data for training a model to predict future flu outbreaks

aws training and certification

Participant ID	Age	Income	Education	State	Flu shot	Outbreak zone?
123456	39	45,000/ year	4 yr degree	New York	Yes	No
123457	23	18,000/ year	HS diploma	Minnesota	No	Yes
123458	78		Masters/PhD		Yes	Yes
123459	20	3,000,000 /year	HS diploma	California	No	No
123460	154	53,000 /year		Masters/PhD		
...	...	...	...	...	...	...

More than one feature in the same column

An even messier example is when you have a column of data that has multiple features represented. For instance, the stat column also has data from other columns in it. In this situation, you have to reshape the data so that each specific feature will be in its own column.

Example data for training a model to predict future flu outbreaks

aws training and certification

Participant ID	Age	Income	Education	State	Flu shot	Outbreak zone?
123456	39	45,000/ year	4 yr degree	New York	Yes	No
123457	23	18,000/ year	HS diploma	Minnesota	No	Yes
123458	78		Masters/PhD		Yes	Yes
123459	20	3,000,000 /year	HS diploma	California	No	No
123460	154	53,000 /year		Masters/PhD		
...	...	...	...	...	...	...

Get the data scientist to normalize this column

To reshape the data so that the features are in their own column, you need the column to be normalized. If you're not familiar with this process that's ok, your data scientist can help.